A 4-approximation for scheduling on a single machine with general cost function

Julián Mestre* José Verschae†

March 4, 2014

Abstract

We consider a single machine scheduling problem that seeks to minimize a generalized cost function: given a subset of jobs we must order them so as to minimize $\sum f_j(C_j)$, where C_j is the completion time of job j and f_j is a job-dependent cost function. This problem has received a considerably amount of attention lately, partly because it generalizes a large number of sequencing problems while still allowing constant approximation guarantees.

In a recent paper, Cheung and Shmoys [6] provided a primal-dual algorithm for the problem and claimed that is a 2-approximation. In this paper we show that their analysis cannot yield an approximation guarantee better than 4. We then cast their algorithm as a local ratio algorithm and show that in fact it has an approximation ratio of 4. Additionally, we consider a more general problem where jobs has release dates and can be preempted. For this version we give a 4κ-approximation algorithm where κ is the number of distinct release dates.

1 Introduction

We study the problem of scheduling jobs on a single machine with generalized min-sum cost function. Consider a set of jobs $\mathcal{J}$ each with a given processing time p_j and an arbitrary non-decreasing non-negative cost function f_j. We aim to find a single machine schedule that minimizes $\sum_j f_j(C_j)$, where C_j denotes the completion time of job j. In the 3-field notation by Graham *et al.* [8] this problem is denoted by $1||\sum f_j$. The problem is strongly NP-hard even in the case where the cost functions are of the form $f_j = w_j \max\{C_j - d_j, 0\}$, which corresponds to *minimizing weighted total tardiness* [10].

A natural variant additionally considers a release date r_j for each job $j \in \mathcal{J}$ and seeks a preemptive schedule with the same min-sum objective. In the case that all release dates are zero, preempting a job cannot help decrease the objective function, and therefore this problem generalizes $1||\sum f_j$. The more general version is denoted by $1|r_j, \text{pmtn}|\sum f_j$.

Bansal and Pruhs [1] were the first to study these two problems: They gave a 16-approximation for $1||\sum f_j$ and an $O(\log\log(n \max_j p_j))$-approximation for $1|r_j, \text{pmtn}|\sum f_j$. For the case without release dates Cheung and Shmoys [6] give a primal-dual algorithm and claim that has an approximation ratio of 2. Their approach is based on a natural time-indexed LP relaxation strengthen with the so called *knapsack-cover* inequalities, introduced by Carr *et al.* [5]. Although their algorithm has pseudo-polynomial running time, with standard techniques they can make the algorithm polynomial by only increasing the approximation ratio in a $1 + \varepsilon$ factor.

Our first result is an example showing that the analysis of Cheung and Shmoys cannot imply an approximation guarantee better than 4. Then we interpret their algorithm in the *local ratio* framework and show that it is in fact 4-approximate. We also give a natural generalization of the

*School of Information Technologies, University of Sydney, Australia. Email:julian.mestre@sydney.edu.au. Supported by an ARC grant DE130101664.

†Departamento de Ingeniería Industrial and Centro de Modelamiento Matemático, Universidad de Chile, Santiago, Chile. Email: jverscha@ing.uchile.cl. Supported by the Nucleo Milenio Información y Coordinación en Redes ICM/FIC P10-024F and by FONDECYT project 3130407.

techniques to $1|r_j, \text{pmtn}|\sum f_j$ that yields a 4κ-approximation algorithm, where κ is the number of distinct release dates. With the same technique of Cheung and Shmoys these algorithms can be made to run in polynomial time by loosing a $1+\varepsilon$ factor in the approximation guarantee.

Previous Work

Besides the previously mentioned results by Bansal and Pruhs [1] and Cheung and Shmoys [6], there is plenty of literature on special cases of the problem.

Epstein et al. [7] study the problem $1||\sum_j w_j f$, corresponding to the case where functions f_j are a scaled versions of a common function f. This problem is equivalent to minimize $\sum_j w_j C_j$ on a machine that changes its speed over time. They give an algorithm that computes a sequence of jobs which simultaneously yields a $(4+\varepsilon)$-approximate schedule for any function f. Using randomization this result can be improved to a $(e+\varepsilon)$-approximation. They also consider the version with release dates and obtain analogous results. However, in this setting a schedule is specified by a priority order used in a preemptive list schedule procedure. Very recently Megow and Verschae [11] consider the version without release dates, $1||\sum_j w_j f$, and obtain a PTAS. This result is best possible since the problem is strongly NP-hard. Höhn and Jacobs [9] study the problem for a convex or concave function f. Their main result is a method to compute the exact approximation ratio of Smith's rule. Additionally, they show that the problem is strongly NP-hard even for piece-wise linear functions f.

In its full generality, $1|r_j, \text{pmtn}|\sum f_j$ models several problems that are far from being understood from an approximation point of view. The most prominent of these problems is *minimizing weighted flow time* $1|r_j|\sum_j w_j(C_j - r_j)$. Here $C_j - r_j$ is the flow time of job j, i.e., the amount of time that the job is alive until it is served. Another related objective is *minimizing squared flow time* $\sum_j (C_j - r_j)^2$, or more generally $\sum_j (C_j - r_j)^k$ for some $k \in \{2, 3, \ldots\}$, which were recently proved to be NP-hard by Moseley *et al.* [12]. On the positive side, the best approximation guarantee for all these problems is the $O(\log\log(n \max_j p_j))$-approximation by Bansal and Pruhs [1]. In principle there is no theoretical reason to rule out the existence of a PTAS, even for the general problem $1|r_j, \text{pmtn}|\sum f_j$. This remains one of the most intriguing questions in this area.

The local ratio technique has been widely used in the design of approximation algorithms [3]. Local-ratio and primal-dual algorithms are closely related; indeed, the two frameworks are known to be equivalent [4]. Our local ratio interpretation of the Cheung-Shmoys algorithm is inspired by the local ratio algorithm for the resource allocation problem studied by Bar-Noy *et al.* [2].

Paper Overview

In Section 2 we review the Cheung-Shmoys algorithm. As mentioned before, the algorithm is based on a time-index LP relaxation. This LP is used in a primal-dual framework that construct simultaneously a primal and a dual solution. Here the primal solution defines a schedule while the dual serves to define a lower bound on the optimal cost. In Section 2.1 we give an instance where the algorithm returns a primal solution whose cost is roughly 4 times the cost of the dual solution constructed. This implies that the primal-dual analysis by Cheung and Shmoys showing an approximation guarantee of 2 is incorrect; in Appendix A we discuss the precise issue with their proof. We must remark, however, that our instance does not rule out that the algorithm could be a 2-approximation. However, it does show that a different proof strategy would be necessary.

Our main result is given in Section 3, where we show that the Cheung-Shmoys is in fact a pseudo-polynomial time 4-approximation. We do this by interpreting their approach in a local ratio framework. Finally, we generalize this algorithm to the setting with release dates in Section 4, obtaining a pseudo-polynomial time 4κ-approximation[1]. Both algorithms can be

[1] The reason of presenting the algorithm in the local ratio framework is twofold. First, we believe that a different interpretation helps shed some light on the techniques of Cheung and Shmoys. Second, it greatly simplifies the notation, specially for the case with release dates.

modified so as to run in polynomial time at the expense of increasing the approximation factor by ε.

2 The Cheung-Shmoys algorithm

Before describing the Cheung-Shmoys algorithm we need to introduce some notation. We assume the processing times p_j are integral. Let $\mathcal{T} = \{1, \ldots, T\}$ be the set of time slots our schedule is allowed to use and $\mathcal{J} = \{1, \ldots, n\}$ be the set of jobs we are to schedule. Note that $T = \sum_{j \in \mathcal{J}} p_j$.

For any given time $t \in \mathcal{T}$ we denote by $D(t) = T - t + 1$ the *demand* at time t. The total processing time of jobs finishing at t or later needs to be at least $D(t)$. Moreover, given $t \in \mathcal{T}$ and a subset of jobs $A \subseteq \mathcal{J}$ we define $D(t, A) = \max\{0, T - t + 1 - p(A)\}$, which corresponds to the demand to be cover at t even if jobs in A finish at t or later. We refer to $D(t, A)$ as the *residual demand* at time t given set A. Similarly, $p_j(t, A) = \min\{p_j, D(t, A)\}$ is the *truncated processing time* of job j at time t given set A. The interpretation of this value is as follows: if all jobs in A are processed at t or later, $p_j(t, A)$ is the maximum amount of residual demand $D(t, A)$ that job j can cover.

The algorithm is based on the following pair of primal and dual linear programs. The primal uses binary variables $x_{j,t}$ that indicate the time t at which job j completes.

$$\begin{aligned}
&\text{minimize} && \sum_{j \in \mathcal{J}} \sum_{t \in \mathcal{T}} f_j(t)\, x_{j,t} && \\
&\text{subject to} && \sum_{j \notin A} \sum_{s \in \mathcal{T} : s \geq t} p_j(t, A)\, x_{j,s} \geq D(t, A) && \text{for all } t \in \mathcal{T} \text{ and } A \subseteq \mathcal{J}, \\
& && x_{j,t} \geq 0 && \text{for all } j \in \mathcal{J} \text{ and } t \in \mathcal{T}, \\
&\text{maximize} && \sum_{t \in \mathcal{T}} \sum_{A \subseteq \mathcal{J}} D(t, A)\, y_{t,A} && \\
&\text{subject to} && \sum_{t \in \mathcal{T} : t \leq s} \sum_{A \subseteq \mathcal{J} : j \notin A} p_j(t, A)\, y_{t,A} \leq f_j(s) && \text{for all } j \in \mathcal{J} \text{ and } s \in \mathcal{T}, \\
& && y_{t,A} \geq 0 && \text{for all } t \in \mathcal{T} \text{ and } A \subseteq \mathcal{J}.
\end{aligned}$$

The pseudo-code of the procedure appears in Algorithm 1. It has a primal growing phase followed by a pruning phase. In the growing phase (Lines 4–9) the algorithm builds a tentative primal solution x and a dual solution y. In the pruning phase (Lines 10–17) the algorithm resets some of the primal variables that are not needed anymore to maintain feasibility. Finally, in the analysis one shows that the cost of the primal solution is at most β times the cost of the dual solution, where β is the targeted approximation factor.

Algorithm 1 CHEUNG-SHMOYS(f, p)

1. $x, y = 0$ // *primal and dual solutions*
2. $A_t = \emptyset$ for each $t \in \mathcal{T}$ // *set of jobs assigned to t or later by x*
3. $k = 1$ // *iteration counter*
4. **while** $\exists t : D(t, A_t) > 0$ **do**
5. $\quad t^k = \text{argmax}_t\, D(t, A_t)$ (break ties by choosing largest time index)
6. $\quad$ Increase $y_{t^k, A_{t^k}}$ until a dual constraint with right-hand side $f_j(t)$ becomes tight (break ties by choosing largest time index)
7. $\quad x_{j,t} = 1$
8. $\quad A_s = A_s \cup \{j\}$ for all $s \in \mathcal{T} : s \leq t$
9. $\quad k = k + 1$
10. **for** $(j, t) : x_{j,t} = 1$, in reverse order in which the variables were set **do**
11. $\quad$ **if** $j \in A_{t+1}$ **then**
12. $\quad\quad x_{j,t} = 0$
13. $\quad$ **else**
14. $\quad\quad S = \{s \leq t : j$ was added to A_s in the same iteration we set $x_{j,t} = 1\}$
15. $\quad\quad$ **if** $\sum_{j' \in A_s \setminus \{j\}} p_{j'} \geq D(s)$ for all $s \in S$ **then**
16. $\quad\quad\quad x_{j,t} = 0$
17. $\quad\quad\quad A_s = A_s \setminus \{j\}$ for all $s \in S$
18. **for** $j \in \mathcal{J}$ **do**
19. $\quad$ set due date d_j to t if $x_{j,t} = 1$
20. Schedule jobs using the EDD rule

2.1 Counterexample

We will show that a primal-dual analysis of CHEUNG-SHMOYS cannot yield an approximation ratio better than 4. Appendix A discusses the specific issue with the argument provided by Cheung and Shmoys [6].

Lemma 1. *For any $\varepsilon > 0$ there exists an instance where the* CHEUNG-SHMOYS *algorithm constructs a pair of primal-dual solutions with a gap of $4 - \varepsilon$.*

Proof. Consider an instance with 4 jobs. Let $p \geq 4$ be an integer. For $j \in \{1, 2, 3, 4\}$, we define the processing times as $p_j = p$ and the cost functions as

$$f_1(t) = f_2(t) = \begin{cases} 0 & \text{if } 1 \leq t \leq p-1, \\ p & \text{if } p \leq t \leq 3p-1, \\ \infty & \text{otherwise, and} \end{cases}$$

$$f_3(t) = f_4(t) = \begin{cases} 0 & \text{if } 1 \leq t \leq 3p-2, \\ p & \text{otherwise.} \end{cases}$$

The following table shows the key variables of the algorithm in each iteration of the growing phase and the corresponding updates to the dual and primal solutions. It is worth noting that the algorithm breaks ties by favoring the largest time index (Lines 5 and 6) and that our example follows this rule.

Iteration k	t^k	$A^k_{t^k}$	$D(t_k, A^k_{t^k})$	Dual update	Primal update
1	1	$\emptyset$	$4p$	$y_{1,\emptyset} = 0$	$x_{3,3p-2} = 1$
2	1	$\{3\}$	$3p$	$y_{1,\{3\}} = 0$	$x_{4,3p-2} = 1$
3	1	$\{3,4\}$	$2p$	$y_{1,\{3,4\}} = 0$	$x_{2,p-1} = 1$
4	$3p-1$	$\emptyset$	$p+2$	$y_{3p-1,\emptyset} = 1$	$x_{4,4p} = 1$
5	p	$\{3,4\}$	$p+1$	$y_{p,\{3,4\}} = 0$	$x_{2,3p-1} = 1$
6	1	$\{2,3,4\}$	p	$y_{p,\{2,3,4\}} = 0$	$x_{1,3p-1} = 1$
7	$3p$	$\{4\}$	1	$y_{3p,\{4\}} = 0$	$x_{3,4p} = 1$

Notice that the only non-zero dual variable the algorithm sets is $y_{3p-1,\emptyset} = 1$. Thus the dual value achieved is $y_{3p-1,\emptyset} D(3p-1,\emptyset) = p+2$. It is not hard to see that the pruning phase keeps the largest due date for each job and has cost $4p$. In fact, it is not possible to obtain a primal (integral) solution with cost less than $4p$: We must pay p for each job 3 and 4 in order to cover the demand at time $3p$, and we must pay p for each job 1 and 2 since they cannot finish before time p. Therefore the pair of primal-dual solutions have a gap of $4p/(p+2)$, which converges to 4 as p tends to infinity. □

The attentive reader would complain that the cost functions used in the proof Lemma 1 are somewhat artificial. Indeed, jobs 1 and 2 cost 0 only in $[0, p-1]$ even though it is not possible to finish them before p. This is, however, not an issue since given any instance (f, p) of the problem we can obtain a new instance (f', p') where $f'_j(t) \geq f'_j(p'_j)$ for all t where we observe essentially the same primal-dual gap in (f, p) and (f', p'). The transformation is as follows: First, we create a dummy job with processing time $T = \sum_j p_j$ that costs 0 up to time T and infinity after that. Second, for each of the original jobs j, we keep their old processing times, $p'_j = p_j$, but modify their cost function:

$$f'_j(t) = \begin{cases} \delta p_j & \text{if } t \geq T, \\ \delta p_j + f_j(t-T) & \text{if } T < t \leq 2T. \end{cases}$$

In other words, to obtain f'_j we shift f_j by T units of time to the right and then add δp_j everywhere, where δ is an arbitrarily small value.

Consider the execution of CHEUNG-SHMOYS on the modified instance (f', p'). In the first iteration, the algorithm sets $y_{1,\emptyset}$ to 0 and assigns the dummy job to time T. In the second iteration, the algorithm chooses to increase the dual variable $y_{T+1,\emptyset}$. Imagine increasing this variable in a continuous way and consider the moment when it reaches δ. At this instant, the slack of the dual constraints for times in $[T+1, 2T]$ in the modified instance are identical to the slack for times in $[1, T]$ at the beginning of the execution on the original instance (f, p). From this point in time onwards, the execution on the modified instance will follow the execution on the original instance but shifted T units of time to the right. The modified instance gains only an extra δT of dual value, which can be made arbitrarily small, so we observe essentially the same primal-dual gap on (f', p') as we do on (f, p).

3 A 4-approximation via local-ratio

In this section we cast the primal-dual algorithm of Cheung and Shmoys as a local-ratio algorithm and prove that it is a pseudo-polynomial time 4-approximation. At the end of the section, we discuss how to turn this algorithm into a polynomial time $(4+\varepsilon)$-approximation.

We will work with due date assignment vectors $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n) \in (\mathcal{T} \cup \{0\})^n$, where $\sigma_j = t$ means that job j has a due date of t. We will use the short-hand notation $(\boldsymbol{\sigma}_{-j}, s)$ to denote the assignment where j is given a due date s and all other jobs get their $\boldsymbol{\sigma}$ due date; that is,

$$(\boldsymbol{\sigma}_{-j}, s) = (\sigma_1, \ldots, \sigma_{j-1}, s, \sigma_{j+1}, \ldots, \sigma_n).$$

We call an assignment $\boldsymbol{\sigma}$ *feasible*, if there is a schedule of the jobs that meets all due dates. We say that job $j \in \mathcal{J}$ *covers* time t if $\sigma_j \geq t$. The cost of $\boldsymbol{\sigma}$ under the cost function vector

Algorithm 2 LR-CS$(\boldsymbol{\sigma}, \mathbf{g})$

if $\boldsymbol{\sigma}$ is feasible **then**
  $\boldsymbol{\rho} = \boldsymbol{\sigma}$
else
  $t^* = \operatorname{argmax}_{t \in \mathcal{T}} D(t, \boldsymbol{\sigma})$ // break ties arbitrarily
  For each $i \in \mathcal{J}$ let $\widehat{g}_i(t) = \begin{cases} p_i(t^*, \boldsymbol{\sigma}) & \text{if } \sigma_i < t^* \leq t, \\ 0 & \text{otherwise} \end{cases}$
  Set $\widetilde{\mathbf{g}} = \mathbf{g} - \alpha \cdot \widehat{\mathbf{g}}$ where α is the largest value such that $\widetilde{\mathbf{g}} \geq 0$
  Let j and s be such that $\widetilde{g}_j(s) = 0$ and $\widehat{g}_j(s) > 0$
  $\widetilde{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}_{-j}, s)$
  $\widetilde{\boldsymbol{\rho}} =$ LR-CS$(\widetilde{\boldsymbol{\sigma}}, \widetilde{\mathbf{g}})$
  if $(\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$ is feasible **then**
    $\boldsymbol{\rho} = (\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$
  else
    $\boldsymbol{\rho} = \widetilde{\boldsymbol{\rho}}$
return $\boldsymbol{\rho}$

$\mathbf{g} = (g_1, \ldots, g_n)$ is defined as $\mathbf{g}(\boldsymbol{\sigma}) = \sum_{j \in \mathcal{J}} g_j(\sigma_j)$. We denote by $A_t^{\boldsymbol{\sigma}} = \{j \in \mathcal{J} : \sigma_j \geq t\}$, the set of jobs that cover t. We call

$$D(t, \boldsymbol{\sigma}) = D(t, A_t^{\boldsymbol{\sigma}}) = \max\{T - t + 1 - p(A_t^{\boldsymbol{\sigma}}), 0\}$$

the *residual demand* at time t with respect to assignment $\boldsymbol{\sigma}$. And

$$p_j(t, \boldsymbol{\sigma}) = p_j(t, A_t^{\boldsymbol{\sigma}}) = \min\{p_j, D(t, \boldsymbol{\sigma})\}$$

the *truncated processing time* of j with respect to t and $\boldsymbol{\sigma}$.

The following is a well-known fact from Scheduling Theory.

Lemma 2. *An assignment $\boldsymbol{\sigma}$ is feasible if there is no residual demand at any time step; namely, if $D(t, \boldsymbol{\sigma}) = 0$ for all $t \in \mathcal{T}$. Furthermore, scheduling the jobs according to earliest due date first (EDD) yields a feasible schedule.*

Proof. For a fix job j, let C_j be its completion time in the EDD schedule and $t = \sigma_j$ its due date. We show that $C_j \leq t$. Consider the set $A = \{i \in \mathcal{J} : \sigma_i \geq \sigma_j + 1\}$ of jobs with due dates at $\sigma_j + 1$ or later. Since $\boldsymbol{\sigma}$ leaves no residual demand at $t + 1$, then $\sum_{i \in A} p_i \geq D(t + 1) = T - t$. Noticing that all jobs in A are processed after j and that the schedule does not leave any idle time, we obtain that $C_j \leq T - \sum_{i \in A} p_i \leq T - (T - t) = t$. □

At a very high level, the algorithm, which we call LR-CS, works as follows: We start by assigning a due date of 0 to all jobs; then we iteratively increase the due dates until the assignment is feasible; finally, we try to undo each increase in reverse order as long as it preserves feasibility.

In the analysis we will argue that the due date assignment that the algorithm ultimately returns is feasible and that the cost of any schedule that meets the due dates is a 4-approximation. Together with Lemma 2 this implies the main result in this section.

Theorem 3. *There is a pseudo-polynomial time 4-approximation algorithm for scheduling jobs on a single machine with generalized cost functions.*

We now describe the algorithm in more detail. Then we prove that is a 4-approximation. For reference, its pseudo-code is given in Algorithm 2.

3.1 Formal description of the algorithm

The algorithm is recursive. It takes as input an assignment vector $\boldsymbol{\sigma}$ and a cost function vector $\mathbf{g}$, and returns a feasible assignment $\boldsymbol{\rho}$. Initially, the algorithm is called on the trivial assignment $(0, \ldots, 0)$ and the instance cost function vector $(f_1, \ldots, f_n)$. As the algorithm progresses, both vectors are modified. We assume, without loss of generality, that $f_j(0) = 0$ for all $j \in \mathcal{J}$.

First, the algorithm checks if the input assignment $\boldsymbol{\sigma}$ is feasible. If that is the case, it returns $\boldsymbol{\rho} = \boldsymbol{\sigma}$. Otherwise, it decomposes the input vector function $\mathbf{g}$ into two cost function vectors $\widetilde{\mathbf{g}}$ and $\widehat{\mathbf{g}}$ as follows

$$\mathbf{g} = \widetilde{\mathbf{g}} + \alpha \cdot \widehat{\mathbf{g}},$$

where α is largest value such that $\widetilde{\mathbf{g}} \geq \mathbf{0}$ [2], and $\widehat{\mathbf{g}}$ will be specified later.

It selects a job j and a time s such that $\widehat{g}_j(s) > 0$ and $\widetilde{g}_j(s) = 0$, and builds a new assignment $\widetilde{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}_{-j}, s)$ thus increasing the due date of j to s while keeping the remaining due dates fixed. It then makes a recursive call LR-CS$(\widetilde{\mathbf{g}}, \widetilde{\boldsymbol{\sigma}})$, which returns a feasible assignment $\widetilde{\boldsymbol{\rho}}$. Finally, it tests the feasibility of reducing the deadline of job j in $\widetilde{\boldsymbol{\rho}}$ back to σ_j. If the resulting assignment is still feasible, it returns that; otherwise, it returns $\widetilde{\boldsymbol{\rho}}$.

The only part that remains to be specified is how to decompose the cost function vector. Let t^* be a time slot with maximum residual unsatisfied demand with respect to $\boldsymbol{\sigma}$:

$$t^* \in \operatorname{argmax}_{t \in \mathcal{T}} D(t, \boldsymbol{\sigma}).$$

The algorithm creates, for each job $i \in \mathcal{J}$, a model cost function

$$\widehat{g}_i(t) = \begin{cases} p_i(t^*, \boldsymbol{\sigma}) & \text{if } \sigma_i < t^* \leq t, \\ 0 & \text{otherwise.} \end{cases}$$

and chooses α to be the largest value such that

$$\widetilde{g}_i(t) = g_i(t) - \alpha \widehat{g}_i(t) \geq 0 \qquad \text{for all } i \in \mathcal{J} \text{ and } t \in \mathcal{T}.$$

In the primal-dual interpretation of the algorithm, α is the value assigned to the dual variable $y(t^*, A^{\sigma}_{t^*})$.

Let (j, s) be a job-time pair that prevented us from increasing α further. In other words, let (j, s) be such that $\widetilde{g}_j(s) = 0$ and $\widehat{g}_j(s) > 0$.

Observation 4. *If $j \in \mathcal{J}$ and $s \in \mathcal{T}$ are such that $\widehat{g}_j(s) > 0$ then $\sigma_j < t^* \leq s$.*

Intuitively, assigning a due date of s to job j helps cover some of the residual demand at t^*. This is precisely what the algorithm does: The assignment used as input for the recursive call is $\widetilde{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}_{-j}, s)$.

3.2 Analysis

For a given vector $\mathbf{g}$ of non-negative functions, we denote by $\operatorname{opt}(\mathbf{g})$ the cost of an optimal schedule under these cost functions. We say an assignment $\boldsymbol{\rho}$ is β-approximate with respect to $\mathbf{g}$ if $\sum_{i \in \mathcal{J}} g_i(\rho_i) \leq \beta \cdot \operatorname{opt}(\mathbf{g})$.

The correctness of the algorithm rests on the following Lemmas.

Lemma 5. *Let $(\boldsymbol{\sigma}^{(1)}, \mathbf{g}^{(1)}), (\boldsymbol{\sigma}^{(2)}, \mathbf{g}^{(2)}), \ldots, (\boldsymbol{\sigma}^{(k)}, \mathbf{g}^{(k)})$ be the inputs to the successive recursive calls to* LR-CS *and let $\boldsymbol{\rho}^{(1)}, \boldsymbol{\rho}^{(2)}, \ldots, \boldsymbol{\rho}^{(k)}$ be their corresponding outputs. The following properties hold:*

(i) $\boldsymbol{\sigma}^{(1)} \leq \boldsymbol{\sigma}^{(2)} \leq \cdots \leq \boldsymbol{\sigma}^{(k)}$,

(ii) $\boldsymbol{\rho}^{(1)} \leq \boldsymbol{\rho}^{(2)} \leq \cdots \leq \boldsymbol{\rho}^{(k)}$,

[2] By $\mathbf{g} = \widetilde{\mathbf{g}} + \alpha \cdot \widehat{\mathbf{g}}$, we mean $g_j(t) = \widetilde{g}_j(t) + \alpha \cdot \widehat{g}_j(t)$ for all $t \in \mathcal{T}$ and $j \in \mathcal{J}$.
By $\widetilde{\mathbf{g}} \geq \mathbf{0}$, we mean $\widetilde{g}_j(t) \geq 0$ for all $j \in \mathcal{J}$, $t \in \mathcal{T}$.

(iii) $\boldsymbol{\sigma^{(i)}} \leq \boldsymbol{\rho^{(i)}}$ *for all* $i = 1, \ldots, k$,

(iv) $g_j^{(i)}(\sigma_j^{(i)}) = 0$ *and* $g_j^{(i)}$ *is non-negative for all* $i = 1, \ldots, k$ *and* $j \in \mathcal{J}$.

Proof. The first property follows from the fact that $\boldsymbol{\sigma^{(i+1)}}$ is constructed by taking $\boldsymbol{\sigma^{(i)}}$ and increasing the due date of a single job.

The second property follows from the fact that $\boldsymbol{\rho^{(i)}}$ is either $\boldsymbol{\rho^{(i+1)}}$ or it is constructed by taking $\boldsymbol{\rho^{(i+1)}}$ and decreasing the due date of a single job.

The third property follows by an inductive argument. The base case is the base case of the recursion, where $\boldsymbol{\sigma^{(k)}} = \boldsymbol{\rho^{(k)}}$. For the recursive case, we need to show that $\boldsymbol{\sigma^{(i)}} \leq \boldsymbol{\rho^{(i)}}$, by recursive hypothesis we know that $\boldsymbol{\sigma^{(i+1)}} \leq \boldsymbol{\rho^{(i+1)}}$ and by the first property $\boldsymbol{\sigma^{(i)}} \leq \boldsymbol{\sigma^{(i+1)}}$. The algorithm either sets $\boldsymbol{\rho^{(i)}} = \boldsymbol{\rho^{(i+1)}}$, or $\boldsymbol{\rho^{(i)}}$ is constructed by taking $\boldsymbol{\rho^{(i+1)}}$ and decreasing the due date of some job to its old $\boldsymbol{\sigma^{(i)}}$ value. In both cases the property holds.

The forth property also follows by induction. The base case is the first call we make to LR-CS, which is $\boldsymbol{\sigma^{(1)}} = (0, \ldots, 0)$ and $\mathbf{g^{(1)}} = (f_1, \ldots, f_n)$, where it holds by our assumption. For the inductive case, we note that $\mathbf{g^{(i+1)}}$ is constructed by taking $\mathbf{g^{(i)}}$ and subtracting a scaled version of the model function vector, so that $\mathbf{0} \leq \mathbf{g^{(i+1)}} \leq \mathbf{g^{(i)}}$, and $\boldsymbol{\sigma^{(i+1)}}$ is constructed by taking $\boldsymbol{\sigma^{(i)}}$ and increasing the due date of a single job $j^{(i)}$. The way this is done guarantees $g_{j^{(i)}}^{(i+1)}(\sigma_{j^{(i)}}^{(i+1)}) = 0$, which ensures that the property holds. □

Lemma 6. *Let* LR-CS$(\boldsymbol{\sigma}, \mathbf{g})$ *be a recursive call returning* $\boldsymbol{\rho}$ *then*

$$\sum_{i \in \mathcal{J} \,:\, \sigma_i < t^* \leq \rho_i} p_i(t^*, \boldsymbol{\sigma}) \leq 4 \cdot D(t^*, \boldsymbol{\sigma}). \tag{1}$$

where t^* *is the value used to decompose the input cost function vector* $\mathbf{g}$.

Proof. Our goal is to bound the $p_i(t^*, \boldsymbol{\sigma})$ value of jobs in

$$X = \{i \in \mathcal{J} : \sigma_i < t^* \leq \rho_i\}.$$

Notice that the algorithm increases the due date of these jobs in this or a later recursive call. Furthermore, and more important to us, the algorithm decides not to undo the increase. For each $i \in X$, consider the call LR-CS$(\boldsymbol{\sigma'}, \mathbf{g'})$ when we first increased the due date of i beyond σ_i. Let $\boldsymbol{\rho'}$ be the assignment returned by the call. Notice that $\rho'_i > \sigma_i$ and that $(\boldsymbol{\rho'}_{-i}, \sigma_i)$ is not feasible—otherwise we would have undone the due date increase. By Lemma 5, we know that $\boldsymbol{\rho} \leq \boldsymbol{\rho'}$, so we conclude that $(\boldsymbol{\rho}_{-i}, \sigma_i)$ is not feasible either. Let t_i be a time with positive residual demand in this unfeasible assignment:

$$D(t_i, (\boldsymbol{\rho}_{-i}, \sigma_i)) > 0.$$

Note that $\sigma_i < t_i \leq \rho_i$, otherwise $\boldsymbol{\rho}$ would not be feasible, contradicting Lemma 5.

We partition X into two subsets

$$L = \{i \in X : t_i \leq t^*\} \text{ and } R = \{i \in X : t_i > t^*\},$$

and we let $t_L = \max\{t_i : i \in L\}$ and i_L be a job attaining this value. Similarly, we let $t_R = \min\{t_i : i \in R\}$ and i_R be a job attaining this value.

We will bound the contribution of each of these sets separately. Our goal will be to prove that

$$\sum_{i \in L - i_L} p_i \leq D(t^*, \boldsymbol{\sigma}), \text{ and} \tag{2}$$

$$\sum_{i \in R - i_R} p_i \leq D(t^*, \boldsymbol{\sigma}). \tag{3}$$

Let us argue (2) first. Since $D\left(t_L, (\boldsymbol{\rho}_{-i_L}, \sigma_{i_L})\right) > 0$, it follows that

$$\begin{aligned} \sum_{i \in \mathcal{J} - i_L : \rho_i \geq t_L} p_i &< T - t_L + 1 \\ \sum_{i \in \mathcal{J} : \sigma_i \geq t_L} p_i + \sum_{i \in \mathcal{J} - i_L : \rho_i \geq t_L > \sigma_i} p_i &< T - t_L + 1 \\ \sum_{i \in \mathcal{J} - i_L : \rho_i \geq t_L > \sigma_i} p_i &< D(t_L, \boldsymbol{\sigma}) \end{aligned}$$

Recall that $\sigma_i < t_i \leq \rho_i$ for all $i \in X$ and that $t_i \leq t_L \leq t^*$ for all $i \in L$. It follows that the sum on the left-hand side of the last inequality contains all jobs in $L - i_L$. Finally, we note that $D(t_L, \boldsymbol{\sigma}) \leq D(t^*, \boldsymbol{\sigma})$ due to the way LR-CS chooses t^*, which gives us (2).

Now let us argue (3). Since $D\left(t_R, (\boldsymbol{\rho}_{-i_R}, \sigma_{i_R})\right) > 0$, it follows that

$$\begin{aligned} \sum_{i \in \mathcal{J} - i_R : \rho_i \geq t_R} p_i &< T - t_R + 1 \\ \sum_{i \in \mathcal{J} : \sigma_i \geq t_R} p_i + \sum_{i \in \mathcal{J} - i_R : \rho_i \geq t_R > \sigma_i} p_i &< T - t_R + 1 \\ \sum_{i \in \mathcal{J} - i_R : \rho_i \geq t_R > \sigma_i} p_i &< D(t_R, \boldsymbol{\sigma}). \end{aligned}$$

Recall that $\sigma_i < t^*$ for all $i \in X$ and that $t^* < t_R \leq t_i \leq \rho_i$ for all $i \in R$. It follows that the sum in the left-hand side of the last inequality contains all jobs in $R - i_R$. Finally, we note that $D(t_R, \boldsymbol{\sigma}) \leq D(t^*, \boldsymbol{\sigma})$ due to the way LR-CS chooses t^*, which gives us (3).

Finally, we note that $p_i(t^*, \boldsymbol{\sigma}) \leq D(t^*, \boldsymbol{\sigma})$ for all $i \in \mathcal{J}$. Therefore,

$$\begin{aligned} \sum_{i \in X} p_i(t^*, \boldsymbol{\sigma}) &\leq \sum_{i \in L - i_L} p_i + p_{i_L}(t^*, \boldsymbol{\sigma}) + \sum_{i \in R - i_R} p_i + p_{i_R}(t^*, \boldsymbol{\sigma}) \\ &\leq 4 \cdot D(t^*, \boldsymbol{\sigma}). \end{aligned}$$

□

Lemma 7. *Let* LR-SC$(\boldsymbol{\sigma}, \mathbf{g})$ *be a recursive call and* $\boldsymbol{\rho}$ *be its output. Then* $\boldsymbol{\rho}$ *is a feasible 4-approximation w.r.t.* $\mathbf{g}$.

Proof. The proof is by induction. The base case corresponds to the base case of the recursion, where we get as input a feasible assignment $\boldsymbol{\sigma}$, and so $\boldsymbol{\rho} = \boldsymbol{\sigma}$. From Lemma 5 we know that $g_i(\sigma_i) = 0$ for all $i \in \mathcal{J}$, and that the cost functions are non-negative. Therefore, the cost of $\boldsymbol{\rho}$ is optimal since

$$\sum_{i \in \mathcal{J}} g_i(\rho_i) = 0.$$

For the inductive case, the cost function vector $\mathbf{g}$ is decomposed into $\widetilde{\mathbf{g}} + \alpha \cdot \widehat{\mathbf{g}}$. Let (j, s) be the pair used to define $\widetilde{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}_{-j}, s)$. Let $\widetilde{\boldsymbol{\rho}}$ be the assignment returned by the recursive call. By inductive hypothesis, we know that $\widetilde{\boldsymbol{\rho}}$ is feasible and 4-approximate w.r.t. $\widetilde{\mathbf{g}}$.

After the recursive call returns, we check the feasibility of $(\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$. If the vector is feasible, we return the modified assignment; otherwise, we return $\widetilde{\boldsymbol{\rho}}$. In either case $\boldsymbol{\rho}$ is feasible.

We claim that $\boldsymbol{\rho}$ is 4-approximate w.r.t. $\widehat{\mathbf{g}}$. Indeed,

$$\sum_{i \in \mathcal{J}} \widehat{g}_i(\rho_i) = \sum_{i \in \mathcal{J} : \sigma_i < t^* \leq \rho_i} p_i(t^*, \boldsymbol{\sigma}) \leq 4 \cdot D(t^*, \boldsymbol{\sigma}) \leq 4 \cdot \mathrm{opt}(\widehat{\mathbf{g}}),$$

where the first inequality follows from Lemma 6 and the last inequality follows from the fact that the cost of any schedule under $\widehat{\mathbf{g}}$ is given by the $p_i(t^*, \boldsymbol{\sigma})$ value of jobs $i \in \mathcal{J}$ with $\sigma_i < t^* \leq \rho_i$, which must have a combined processing time of at least $D(t^*, \boldsymbol{\sigma})$ on any feasible schedule. Hence, $\mathrm{opt}(\widehat{\mathbf{g}}) \geq D(t^*, \boldsymbol{\sigma})$.

We claim that $\boldsymbol{\rho}$ is 4-approximate w.r.t. $\widetilde{\mathbf{g}}$. Recall that $\widetilde{\boldsymbol{\rho}}$ is 4-approximate w.r.t. $\widetilde{\mathbf{g}}$; therefore, if $\boldsymbol{\rho} = \widetilde{\boldsymbol{\rho}}$ then $\boldsymbol{\rho}$ is 4-approximate w.r.t. $\widetilde{\mathbf{g}}$. Otherwise, $\boldsymbol{\rho} = (\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$, in which case $\widetilde{g}_j(\rho_j) = 0$, so $\boldsymbol{\rho}$ is also 4-approximate w.r.t. $\widetilde{\mathbf{g}}$.

At this point we can invoke the Local Ratio Theorem to get that

$$\begin{aligned}\sum_{j\in\mathcal{J}} g_j(\rho_j) &= \sum_{j\in\mathcal{J}} \widetilde{g}_j(\rho_j) + \sum_{j\in\mathcal{J}} \alpha\cdot\widehat{g}_j(\rho_j),\\ &\leq 4\cdot\operatorname{opt}(\widetilde{\mathbf{g}}) + 4\alpha\cdot\operatorname{opt}(\widehat{\mathbf{g}}),\\ &= 4\cdot\big(\operatorname{opt}(\widetilde{\mathbf{g}}) + \operatorname{opt}(\alpha\cdot\widehat{\mathbf{g}})\big),\\ &\leq 4\cdot\operatorname{opt}(\mathbf{g}),\end{aligned}$$

which finishes the proof of the lemma. □

Note that the number of recursive calls in Algorithm 2 is at most $|\mathcal{J}|\cdot|\mathcal{T}|$. Indeed, by Observation 4 in each call the due date of some job is increased. Therefore we can only guarantee a pseudo-polynomial running time. To obtain a polynomial time algorithm we can use the exact same technique as Cheung and Shmoys and reduce the number of time slots by loosing a factor of $1+\varepsilon$. This is done by considering time intervals in which some cost function increases by a $1+\varepsilon$ factor. This yields an algorithm with running time a polynomial in $|\mathcal{J}|$, $1/\varepsilon$, $\log(\max_j p_j)$, and $\log(\max_j f_j(T))$.

4 Release dates

This section discusses how to generalize the ideas from the previous section to instances with release dates. We assume that there are κ different release dates. Our main result is a pseudo-polynomial 4κ-approximation algorithm. The generalization is surprisingly easy: We only need to re-define our residual demand function to take into account release dates.

For a given due date assignment vector $\boldsymbol{\sigma}$ and an interval $[r,t)$ we denote by

$$D(r,t,\boldsymbol{\sigma}) = \max\left\{r + p\left(\{j\in\mathcal{J} : r\leq r_j\leq\sigma_j<t\}\right) - t + 1, 0\right\}$$

the *residual demand* for $[r,t)$. Intuitively, this quantity is the amount of processing time of jobs released in $[r,t)$ that currently have a due date strictly less than t that should be assigned a due date of t or greater if we want feasibility.

The *truncated processing time* of j with respect to r, t, and $\boldsymbol{\sigma}$ is

$$p_j(r,t,\boldsymbol{\sigma}) = \min\left\{p_j, D(r,t,\boldsymbol{\sigma})\right\}.$$

The algorithm for multiple release dates is very similar to Algorithm 2. The *only* difference is in the way we decompose the input cost function vector $\mathbf{g}$. First, we find values r^* and t^* maximizing $D(r^*,t^*,\boldsymbol{\sigma})$. Second, we define the model cost function for job each $i\in\mathcal{J}$ as follows

$$\widehat{g}_i(t) = \begin{cases} p_i(r^*,t^*,\boldsymbol{\sigma}) & \text{if } r^*\leq r_i<t^* \text{ and } \sigma_i<t^*\leq t,\\ 0 & \text{otherwise.}\end{cases}$$

Algorithm 3 LR-CS-RD($\boldsymbol{\sigma}, \mathbf{g}$)

if $\boldsymbol{\sigma}$ is feasible **then**
  $\boldsymbol{\rho} = \boldsymbol{\sigma}$
else
  $(t^*, r^*) = \text{argmax}_{(t,r)\in\mathcal{T}\times R} D(r, t, \boldsymbol{\sigma})$ // break ties arbitrarily
  For each $i \in \mathcal{J}$ let $\widehat{g}_i(t) = \begin{cases} p_i(r^*, t^*, \boldsymbol{\sigma}) & \text{if } r^* \le r_i < t^* \text{ and } \sigma_i < t^* \le t, \\ 0 & \text{otherwise.} \end{cases}$
  Set $\widetilde{\mathbf{g}} = \mathbf{g} - \alpha \cdot \widehat{\mathbf{g}}$ where α is the largest value such that $\widetilde{\mathbf{g}} \ge 0$
  Let j and s be such that $\widetilde{g}_j(s) = 0$ and $\widehat{g}_j(s) > 0$
  $\widetilde{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}_{-j}, s)$
  $\widetilde{\boldsymbol{\rho}} =$ LR-CS-RD$(\widetilde{\boldsymbol{\sigma}}, \widetilde{\mathbf{g}})$
  if $(\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$ is feasible **then**
    $\boldsymbol{\rho} = (\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$
  else
    $\boldsymbol{\rho} = \widetilde{\boldsymbol{\rho}}$
return $\boldsymbol{\rho}$

The rest of the algorithm is exactly as before. We call the new algorithm LR-CS-RD. Its pseudocode is given in Algorithm 3. The initial call to the algorithm is done on the assignment vector $(r_1, r_2, \ldots, r_n)$ and the function cost vector $(f_1, f_2, \ldots, f_n)$. Without loss of generality, we assume $f_j(r_j) = 0$ for all $j \in \mathcal{J}$.

Theorem 8. *There is a pseudo-polynomial time 4κ-approximation for scheduling jobs with release dates on a single machine with generalized cost function.*

The proof of this theorem rests on a series of Lemmas that mirror Lemmas 2, 5, 6, and 7.

Lemma 9. *An assignment $\boldsymbol{\sigma}$ is feasible if there is no residual demand at any interval $[r, t)$; namely, $\boldsymbol{\sigma}$ is feasible if $D(r, t, \boldsymbol{\sigma}) = 0$ for all $r \in R$ and $r < t \in \mathcal{T}$. Furthermore, scheduling the jobs according to early due date first yields a feasible preemptive schedule.*

Proof. We start by noting that one can use a simple exchange argument to show that if there is some schedule that meets the dealines $\boldsymbol{\sigma}$, then the earliest due date (EDD) schedule must be feasible.

First, we show that if there is a job j in the EDD schedule that does not meet its deadline, then there is an interval $[r, t)$ such that $D(r, t, \boldsymbol{\sigma}) > 0$. Let $t = \sigma_j + 1$ and let $r < t$ be latest point in time that the machine was idle in the EDD schedule. Let $X = \{i \in \mathcal{J} : r \le r_i, \sigma_i < t\}$. Clearly, $r + p(X) \ge t$, otherwise j would have met its due date. Therefore,

$$\begin{aligned} 0 &< r + p(X) - t + 1 \\ &= r + p\left(\{i \in \mathcal{J} : r \le r_i \le \sigma_i < t\}\right) - t + 1 \\ &\le D(r, t, \boldsymbol{\sigma}). \end{aligned}$$

Second, we show that if an interval $[r, t)$ such that $D(r, t, \boldsymbol{\sigma}) > 0$, then there is a job j in the EDD schedule that does not meet its deadline. Let $X = \{i \in \mathcal{J} : r \le r_i, \sigma_i < t\}$. Then,

$$0 < D(r, t, \boldsymbol{\sigma}) = r + p(X) - t + 1 \quad \Longrightarrow \quad r + p(X) \ge t.$$

Let j be the job in X with the largest completion time in the EDD schedule. Notice that the completion time of j is at least $r + p(X) \ge t$. On the other hand, its due date is $\sigma_j < t$. Therefore, the EDD schedule missing j's due date. □

Lemma 10. *Let $(\boldsymbol{\sigma}^{(1)}, \mathbf{g}^{(1)}), (\boldsymbol{\sigma}^{(2)}, \mathbf{g}^{(2)}), \ldots, (\boldsymbol{\sigma}^{(k)}, \mathbf{g}^{(k)})$ be the inputs to the successive recursive calls to* LR-CS-RD *and let $\boldsymbol{\rho}^{(1)}, \boldsymbol{\rho}^{(2)}, \ldots, \boldsymbol{\rho}^{(k)}$ be their corresponding outputs. The following properties hold:*

(i) $\boldsymbol{\sigma^{(1)}} \leq \boldsymbol{\sigma^{(2)}} \leq \cdots \leq \boldsymbol{\sigma^{(k)}}$,

(ii) $\boldsymbol{\rho^{(1)}} \leq \boldsymbol{\rho^{(2)}} \leq \cdots \leq \boldsymbol{\rho^{(k)}}$,

(iii) $\boldsymbol{\sigma^{(i)}} \leq \boldsymbol{\rho^{(i)}}$ *for all* $i = 1, \ldots, k$,

(iv) $g_j^{(i)}(\sigma_j^{(i)}) = 0$ *and* $g_j^{(i)}$ *is non-negative for all* $i = 1, \ldots, k$ *and* $j \in \mathcal{J}$.

Proof. Properties (i)-(iii) follows from the exactly same reasoning as in Lemma 5.

The forth property follows by induction. The base case is the first call we make to LR-CS-RD, which is $\boldsymbol{\sigma^{(1)}} = (r_1, \ldots, r_n)$ and $\mathbf{g^{(1)}} = (f_1, \ldots, f_n)$, where it holds by our assumption. For the inductive case, we note that $\mathbf{g^{(i+1)}}$ is constructed by taking $\mathbf{g^{(i)}}$ and subtracting a scaled version of the model function vector, so that $\mathbf{0} \leq \mathbf{g^{(i+1)}} \leq \mathbf{g^{(i)}}$, and $\boldsymbol{\sigma^{(i+1)}}$ is constructed by taking $\boldsymbol{\sigma^{(i)}}$ and increasing the due date of a single job $j^{(i)}$. The way this is done guarantees $g_{j^{(i)}}^{(i+1)}(\sigma_{j^{(i)}}^{(i+1)}) = 0$, which ensures that the property holds. □

Lemma 11. *Let* LR-CS-RD$(\boldsymbol{\sigma}, \mathbf{g})$ *be a recursive call returning* $\boldsymbol{\rho}$ *then*

$$\sum_{\substack{i \in \mathcal{J} \\ r^* \leq r_i \leq \sigma_i < t^* \leq \rho_i}} p_i(r^*, t^*, \boldsymbol{\sigma}) \leq 4\kappa \cdot D(r^*, t^*, \boldsymbol{\sigma}).$$

where (r^*, t^*) *are the values used to decompose the input cost function vector* $\mathbf{g}$.

Proof. Our goal is to bound the $p_i(r^*, t^*, \boldsymbol{\sigma})$ value of jobs

$$X = \{i \in \mathcal{J} : r \leq r_i \leq \sigma_i < t^* \leq \rho_i\}.$$

Notice that the algorithm increases the due date of these jobs in this or a later recursive call. Furthermore, and more important to us, the algorithm decides not to undo the increase.

For each $i \in X$, consider the call LR-CS-RD$(\boldsymbol{\sigma'}, \mathbf{g'})$ when we first increased the due date of i beyond σ_i. Let $\boldsymbol{\rho'}$ be assignment returned by the call. Notice that $\rho'_i > \sigma_i$ and that $(\boldsymbol{\rho'}_{-i}, \sigma_i)$ is not feasible—otherwise we would have undone the due date increase. By Lemma 5, we know that $\boldsymbol{\rho} \leq \boldsymbol{\rho'}$, so we conclude that $(\boldsymbol{\rho}_{-i}, \sigma_i)$ is not feasible either. We define $r(i) \leq r_j$ and $\sigma_i < t(i) \leq \rho_i$ such that the interval $[r(i), t(i))$ has a positive residual demand in this unfeasible assignment:

$$D(r(i), t(i), (\boldsymbol{\rho}_{-i}, \sigma_i)) > 0.$$

Note that such an interval must exist, otherwise $\boldsymbol{\rho}$ would not be feasible.

We partition X in 2κ subsets. For each release date r we define

$$L(r) = \{i \in X : t(i) \leq t^*, r(i) = r\} \text{ and } R(r) = \{i \in X : t(i) > t^*, r(i) = r\},$$

Let $t_L^r = \max\{t(i) : i \in L(r)\}$ and i_L^r be a job attaining this value. Similarly, consider $t_R^r = \min\{t(i) : i \in R(r)\}$ and i_R^r be a job attaining this value.

We will bound the contribution of each of these sets separately. Our goal will be to prove that for each release date r we have

$$\sum_{i \in L(r) - i_L^r} p_i \leq D(r^*, t^*, \boldsymbol{\sigma}), \text{ and} \tag{4}$$

$$\sum_{i \in R(r) - i_R^r} p_i \leq D(r^*, t^*, \boldsymbol{\sigma}). \tag{5}$$

Let us argue (4) first. Assume $L(r) \neq \emptyset$, so t_L^r is well defined; otherwise, the claim is trivial.

Since $D\left(r, t_L^r, (\boldsymbol{\rho}_{-i_L^r}, \sigma_{i_L^r})\right) > 0$, it follows that

$$\sum_{\substack{i \in \mathcal{J} - i_L^r \\ r \le r_i < t_L^r \le \rho_i}} p_i < r + \sum_{\substack{i \in \mathcal{J} \\ r \le r_i < t_L^r}} p_i - t_L^r + 1$$

$$\sum_{\substack{i \in \mathcal{J} \\ r \le r_i < t_L^r \le \sigma_i}} p_i + \sum_{\substack{i \in \mathcal{J} - i_L^r \\ r \le r_i \le \sigma_i < t_L^r \le \rho_i}} p_i < r + \sum_{\substack{i \in \mathcal{J} \\ r \le r_i < t_L^r}} p_i - t_L^r + 1$$

$$\sum_{\substack{i \in \mathcal{J} - i_L^r \\ r \le r_i \le \sigma_i < t_L^r \le \rho_i}} p_i < D(r, t_L^r, \boldsymbol{\sigma}).$$

Recall that $\sigma_i < t(i)$ for all $i \in X$. Furthermore, $t(i) \le t_L^r$, and thus $\sigma_i < t_L^r$, for all $i \in L(r)$. Also, $t(i) \le \rho_i$ for all $i \in X$. Therefore the sum in the left-hand side of the last inequality contains all jobs in $L(r) - i_L^r$. Finally, we note that $D(r, t_L, \boldsymbol{\sigma}) \le D(r^*, t^*, \boldsymbol{\sigma})$ due to the way LR-CS-RD chooses r^* and t^*, which gives us (4).

Let us argue (5). Assume $R(r) \neq \emptyset$, so t_R^r is well defined; otherwise, the claim is trivial. Since $D\left(r, t_R^r, (\boldsymbol{\rho}_{-i_R^r}, \sigma_{i_R^r})\right) > 0$, it follows that

$$\sum_{\substack{i \in \mathcal{J} - i_R^r \\ r \le r_i < t_R^r \le \rho_i}} p_i < r + \sum_{\substack{i \in \mathcal{J} \\ r \le r_i < t_R^r}} p_i - t_R^r + 1$$

$$\sum_{\substack{i \in \mathcal{J} \\ r \le r_i < t_R^r \le \sigma_i}} p_i + \sum_{\substack{i \in \mathcal{J} - i_R^r \\ r \le r_i \le \sigma_i < t_R^r \le \rho_i}} p_i < r + \sum_{\substack{i \in \mathcal{J} \\ r \le r_i < t_R^r}} p_i - t_R^r + 1$$

$$\sum_{\substack{i \in \mathcal{J} - i_R^r \\ r \le r_i \le \sigma_i < t_R^r \le \rho_i}} p_i < D(r, t_R^r, \boldsymbol{\sigma})$$

Recall that $t(i) \le \rho_i$ for all $i \in X$. Furthermore, $t_R^r \le t(i)$, and thus $t_R^r \le \rho_i$, for all $i \in R(r)$. Also, $t_i > \sigma_i$ for all $i \in X$. Therefore, the sum in the left-hand side of the last inequality contains all jobs in $R(r) - i_R^r$. Finally, we note that $D(r, t_R^r, \boldsymbol{\sigma}) \le D(r^*, t^*, \boldsymbol{\sigma})$ due to the way LR-CS chooses r^* and t^*, which gives us (5).

Finally, we note that $p_i(r^*, t^*, \boldsymbol{\sigma}) \le D(r^*, t^*, \boldsymbol{\sigma})$ for all $i \in \mathcal{J}$. Therefore,

$$\begin{aligned}
\sum_{i \in \mathcal{J} : \rho_i \ge t^*} p_i(r^*, t^*, \boldsymbol{\sigma}) &= \sum_{i \in X} p_i(r^*, t^*, \boldsymbol{\sigma}) \\
&= \sum_r \left(\sum_{i \in L(r)} p_i(r^*, t^*, \boldsymbol{\sigma}) + \sum_{i \in R(r)} p_i(r^*, t^*, \boldsymbol{\sigma}) \right) \\
&\le \sum_r \Big(2 \cdot D(r^*, t^*, \boldsymbol{\sigma}) + 2 \cdot D(r^*, t^*, \boldsymbol{\sigma}) \Big) \\
&= 4\kappa \cdot D(r^*, t^*, \boldsymbol{\sigma}).
\end{aligned}$$

□

Lemma 12. *Let* LR-SC-RD$(\boldsymbol{\sigma}, \mathbf{g})$ *be a recursive call and* $\boldsymbol{\rho}$ *be its output. Then* $\boldsymbol{\rho}$ *is a feasible* 4κ*-approximation w.r.t.* $\mathbf{g}$.

Proof. The proof is by induction. The base case corresponds to the base case of the recurrence where we get as input a feasible assignment $\boldsymbol{\sigma}$, and so $\boldsymbol{\rho} = \boldsymbol{\sigma}$. From Lemma 5 we know that $g_i(\sigma_i) = 0$ for all $i \in \mathcal{J}$, and that the cost functions are non-negative. Therefore, the cost of $\boldsymbol{\rho}$ is optimal since

$$\sum_{i \in \mathcal{J}} g_i(\rho_i) = 0.$$

For the inductive case, the cost function vector $\mathbf{g}$ is decomposed into $\widetilde{\mathbf{g}} + \alpha \cdot \widehat{\mathbf{g}}$. Let (j, s) be the pair used to define $\widetilde{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}_{-j}, s)$. Let $\widetilde{\boldsymbol{\rho}}$ be the assignment returned by the recursive call. By induction hypothesis, we know that $\widetilde{\boldsymbol{\rho}}$ is feasible and 4κ-approximate w.r.t. $\widetilde{\mathbf{g}}$.

After the recursive call returns, we check the feasibility of $(\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$. If the vector is feasible, then we return the modified assignment; otherwise, we return $\widetilde{\boldsymbol{\rho}}$. In either case $\boldsymbol{\rho}$ is feasible.

We claim that $\boldsymbol{\rho}$ is 4κ-approximate w.r.t. $\widehat{\mathbf{g}}$. Indeed,

$$\sum_{i \in \mathcal{J}} \widehat{g}_i(\rho_i) = \sum_{\substack{i \in \mathcal{J} \\ r^* \leq r_i < t^* \leq \rho_i}} p_i(r^*, t^*, \boldsymbol{\sigma}) \leq 4\kappa \cdot D(r^*, t^*, \boldsymbol{\sigma}) \leq 4\kappa \cdot \mathrm{opt}(\widehat{\mathbf{g}}),$$

where the first inequality follows from Lemma 6 and the last inequality follows from the fact that the cost of any schedule under $\widehat{\mathbf{g}}$ is given by the $p_i(r^*, t^*, \boldsymbol{\sigma})$ value of jobs $i \in \mathcal{J}$ with $r^* \leq r_i < t^*$ and $\sigma_i < t^*$ that cover t^*, which must have a combined processing time of at least $D(r^*, t^*, \boldsymbol{\sigma})$. Hence, $\mathrm{opt}(\widehat{\mathbf{g}}) \geq D(r^*, t^*, \boldsymbol{\sigma})$.

We claim that $\boldsymbol{\rho}$ is 4κ-approximate w.r.t. $\widetilde{\mathbf{g}}$. Recall that $\widetilde{\boldsymbol{\rho}}$ is 4κ-approximate w.r.t. $\widetilde{\mathbf{g}}$; therefore, if $\boldsymbol{\rho} = \widetilde{\boldsymbol{\rho}}$ then $\boldsymbol{\rho}$ is 4κ-approximate w.r.t. $\widetilde{\mathbf{g}}$. Otherwise, $\boldsymbol{\rho} = (\widetilde{\boldsymbol{\rho}}_{-j}, \sigma_j)$, in which case $\widetilde{g}_j(\rho_j) = 0$, so $\boldsymbol{\rho}$ is also 4-approximate w.r.t. $\widetilde{\mathbf{g}}$.

At this point we can invoke the Local Ratio Theorem to get that

$$\begin{aligned}
\sum_{j \in \mathcal{J}} g_j(\rho_j) &= \sum_{j \in \mathcal{J}} \widetilde{g}_j(\rho_j) + \sum_{j \in \mathcal{J}} \alpha \cdot \widehat{g}_j(\rho_j), \\
&\leq 4\kappa \cdot \mathrm{opt}(\widetilde{\mathbf{g}}) + 4\kappa \cdot \alpha \cdot \mathrm{opt}(\widehat{\mathbf{g}}), \\
&= 4\kappa \cdot \big(\mathrm{opt}(\widetilde{\mathbf{g}}) + \mathrm{opt}(\alpha \cdot \widehat{\mathbf{g}}) \big), \\
&\leq 4\kappa \cdot \mathrm{opt}(\mathbf{g}),
\end{aligned}$$

which finishes the proof of the lemma. □

References

[1] N. Bansal and K. Pruhs. The geometry of scheduling. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science*, pages 407–414, 2010.

[2] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48:1069–1090, 2001.

[3] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: A unified framework for approximation algorithms in memoriam: Shimon Even 1935-2004. *ACM Computing Surveys*, 36:422–463, 2004.

[4] R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM J. Discrete Math.*, 19:762–797, 2005.

[5] R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 106–115, 2000.

[6] M. Cheung and D. B. Shmoys. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In *Proceedings of the 14th International Workshop on Approximation, Randomization, and Combinatorial Optimization*, pages 135–146, 2011.

[7] L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie. Universal sequencing on an unreliable machine. *SIAM Journal on Computing*, 41:565–586, 2012.

[8] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 4:287–326, 1979.

[9] W. Höhn and T. Jacobs. On the performance of Smith's rule in single-machine scheduling with nonlinear cost. In *Proceeding of the 10th Latin American Symposium on Theoretical Informatics*, pages 482–493, 2012.

[10] E. L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 331–342. 1977.

[11] N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming*, pages 745–756, 2013.

[12] B. Moseley, K. Pruhs, and C. Stein. The complexity of scheduling for p-norms of flow and stretch. In *Proceedings of the 16th Conference on Integer Programming and Combinatorial Optimization*, pages 278–289, 2013.

A The issue with the analysis of Cheung-Shmoys

The analysis of Cheung and Shmoys hinges on the following claim [6, Lemma 5]:

$$\sum_{j \in \bar{A}_t \setminus A} p_j(t, A) \leq 2D(t, A) \qquad \text{for each } (t, A) : y_{t,A} > 0, \tag{6}$$

where $\bar{A}_t = \{j \in \mathcal{J} : d_j \geq t\}$ is the set of jobs assigned a due date greater than or equal to t (at the end of the pruning phase).

Notice that in the example from Section 2.1 this property does not hold for $t = 3p - 1$ and $A = \emptyset$. Indeed, $y_{3p-1,\emptyset} > 0$ and in the final solution all jobs are assigned to $3p - 1$ or later, and thus,

$$\sum_{j \in \bar{A}_{3p-1}} p_j(3p - 1, \emptyset) = 4p,$$

while on the other hand $D(3p - 1, \emptyset) = p + 2$.

Let A_t^k denote the set A_t at the beginning of the k-th iteration of the growing phase. Property (6) is stated as Lemma 5 in [6]. Its proof aims to show that each time t^k is critical with respect to $\bar{x}$ and $A_{t_k}^k$, meaning there exists a job $\ell \in \bar{A}_{t_k} \setminus A_{t_k}^k$ such that $\sum_{j \in \bar{A}_{t_k} \setminus (A_{t_k}^k \cup \{\ell\})} p_j < D(t^k, A_{t_k}^k)$. Notice that this is not in the case in our example for $t^k = 3p - 1$, where $A_{t_k}^k = \emptyset$ and $\bar{A}_{t_k} = \{1, 2, 3, 4\}$.

Looking further in the proof, we see that the critical nature of t^k is shown by contradiction, saying that if t^k is not critical then for each job $\ell \in \bar{A}_{t^k} \setminus A_{t_k}^k$, then there must exists a t_ℓ such that job ℓ is critical with respect to t_ℓ and $A_{t_\ell}^k$, meaning $\sum_{j \in \bar{A}_{t_\ell} \setminus (A_{t_\ell}^k \cup \{\ell\})} p_j < D(t_\ell, A_{t_\ell}^k)$. Two cases are considered: In Case 1 there exists ℓ such that $t_\ell < t^k$ and in Case 2 we have $t_\ell > t^k$ for all ℓ. In our example, we see that for $t^k = 3p - 1$, job $\ell = 1$ has $t_\ell = 1$. Since $t_\ell < t^k$ that puts us in Case 1 of the proof. For this case the proof claims that

$$\bar{A}_{t_\ell} \setminus (A_{t_\ell}^k \cup \{\ell\}) \supseteq \bar{A}_{t^k} \setminus (A_{t^k}^k \cup \{\ell\}).$$

However, this is not true in our example, since

$$\bar{A}_{t_\ell} \setminus (A_{t_\ell}^k \cup \{\ell\}) = \{1, 2, 3, 4\} \setminus (\{2, 3, 4\} \cup \{1\}) = \emptyset,$$

and

$$\bar{A}_{t^k} \setminus (A_{t^k}^k \cup \{\ell\}) = \{1, 2, 3, 4\} \setminus \{1\} = \{2, 3, 4\}.$$

The issue is that jobs 3 and 4 are in $A_{t_\ell}^k \setminus A_{t^k}^k$, and they finally belong to $\bar{A}_{t^k} \setminus A_{t^k}^k$.